BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: 0 462 917 A2

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 91480082.6

(51) Int. Cl.[5]: G06F 11/10

(22) Date of filing: 22.05.91

(30) Priority: 21.06.90 US 542216

(43) Date of publication of application:
27.12.91 Bulletin 91/52

(84) Designated Contracting States:
DE FR GB IT

(71) Applicant: International Business Machines Corporation
Old Orchard Road
Armonk, N.Y. 10504 (US)

(72) Inventor: Bond, Milton Fredrick
1520 16 1/2 Avenue N.W.
Rochester, Minnesota 55901 (US)
Inventor: Clark, Brian Eldridge
6810 Woodbine Court S.E.
Rochester, Minnesota 55904 (US)
Inventor: McRoberts, Raymond Spencer
1907 Oak Knoll Lane N.W.
Rochester, Minnesota 55901 (US)

(74) Representative: Vekemans, André
Compagnie IBM France Département de Propriété Intellectuelle
F-06610 La Gaude (FR)

(54) Method and apparatus for recovering parity protected data.

(57) A storage management mechanism resident on a storage controller (103) maintains parity records on the storage units (121-124) it services. The storage management mechanism includes a status map (106) indicating, for each data block (131-138), the location of the corresponding parity block (131,136), and the status of the data block. If a single storage unit fails, the system continues to operate, and the storage management mechanism is placed in a failure operating mode. While in failure operating mode, the storage management mechanism checks the status map before accessing data on the failed storage unit. If the data has not yet been reconstructed, storage management first reconstructs the data in that block of storage by successively reading and accumulating an Exclusive-OR (108) of the same blocks on all storage units in the parity group, including the parity block. The block of reconstructed data is stored in the location of the parity block, and the status map is updated to indicate that the block has been reconstructed. Once the data has been reconstructed, it is only necessary to read from or write to the former parity block directly. In the same manner, storage management will reconstruct a block of storage on the failed unit before writing to any corresponding block on a non-failed unit, if the block has not yet been reconstructed. In an alternate embodiment, spare areas of storage in the non-failing storage units are allocated to the reconstructed data. The total of these spare areas constitute a virtual spare storage unit. As data is reconstructed, it is placed in the virtual spare unit, and parity is maintained in the normal fashion.

EP 0 462 917 A2

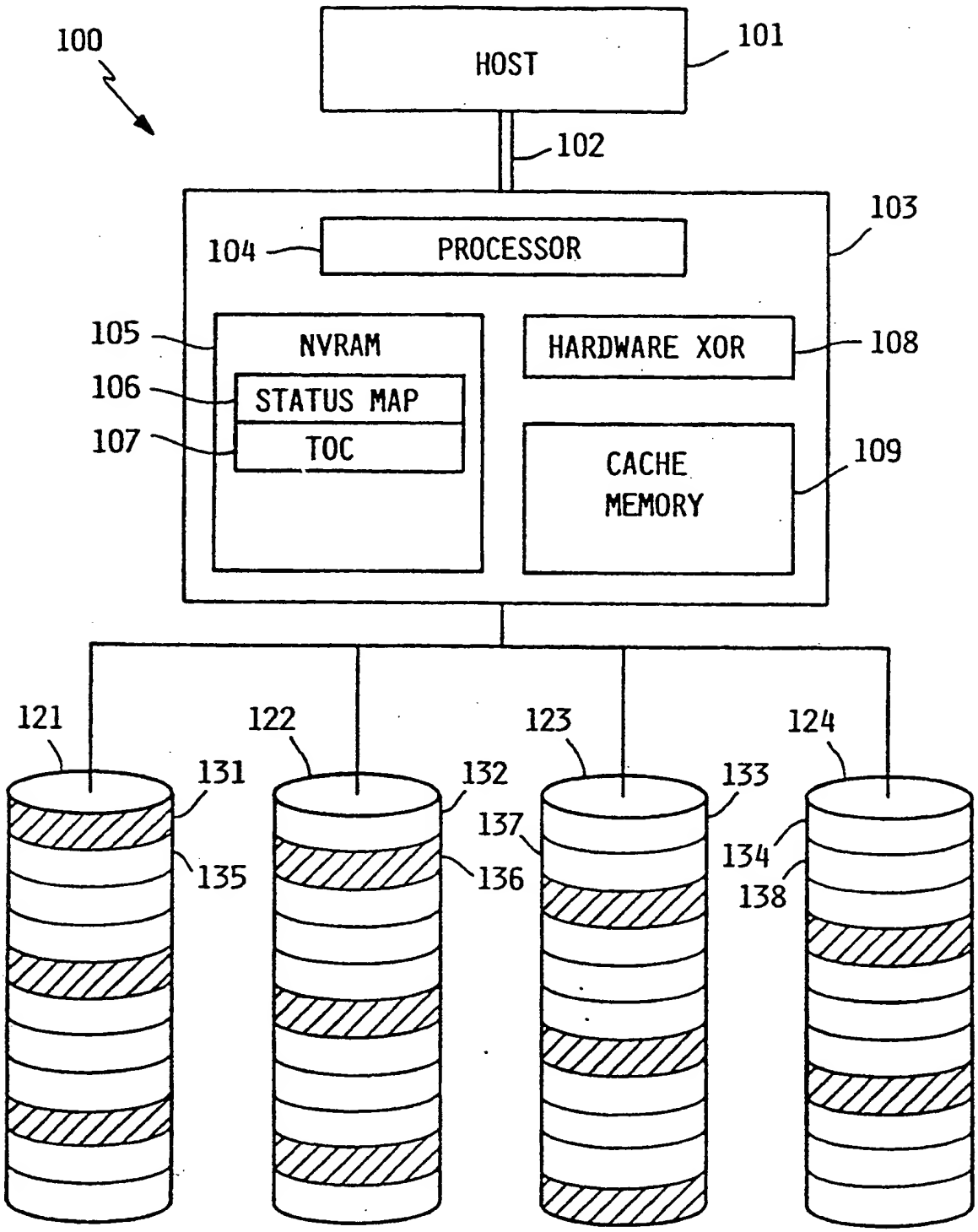
100
HOST
101
102
103
104
PROCESSOR
105
NVRAM
106
STATUS MAP
107
TOC
HARDWARE XOR
108
CACHE MEMORY
109
121
122
123
124
131
132
133
135
136
137
134
138

FIG. 1

The present invention relates to maintaining parity information on computer data storage devices and in particular to maintaining availability of a computer system when reconstructing data from a failed storage device.

The extensive data storage needs of modern computer systems require large capacity mass data storage devices. A common storage device is the magnetic disk drive, a complex piece of machinery containing many parts which are susceptible to failure. A typical computer system will contain several such units. As users increase their need for data storage, systems are configured with larger numbers of storage units. The failure of a single storage unit can be a very disruptive event for the system. Many systems are unable to operate until the defective unit is repaired or replaced, and the lost data restored. An increased number of storage units increases the probability that any one unit will fail, leading to system failure. At the same time, computer users are relying more and more on the consistent availability of their systems. It therefore becomes essential to find improved methods of reconstructing data contained on a failing storage unit, and sustaining system operations in the presence of a storage unit failure.

One method of addressing these problems is known as "mirroring". This method involves maintaining a duplicate set of storage devices, which contains the same data as the original. The duplicate set is available to assume the task of providing data to the system should any unit in the original set fail. Although very effective, this is a very expensive method of resolving with the problem since a customer must pay for twice as many storage devices.

A less expensive alternative is the use of parity blocks. Parity blocks are records formed from the Exclusive-OR of all data records stored at a particular location on different storage units. In other words, each bit in a block of data at a particular location on a storage unit is Exclusive-ORed with every other bit at that same location in each storage unit in a group of units to produce a block of parity bits; the parity block is then stored at the same location on another storage unit. If any storage unit in the group fails, the data contained at any location on the failing unit can be regenerated by taking the Exclusive-OR of the data blocks at the same location on the remaining devices and their corresponding parity block.

U.S. Pat. No. 4,092,732 describes a parity block method. In said patent, a single storage unit is used to store parity information for a group of storage devices. A read and a write on the storage unit containing parity blocks occurs each time a record is changed on any of the storage units in the group covered by the parity record. Thus, the storage unit with the parity records becomes a bottleneck to storage operations. U.S. Patent No. 4,761,785 improves upon storage of parity information by distributing parity blocks substantially equally among a set of storage units. N storage units in a set are divided into a multiple of equally sized address blocks, each containing a plurality of records. Blocks from each storage unit having the same address ranges form a stripe of blocks. Each stripe has a block on one storage device containing parity for the remaining blocks of the stripe. The parity blocks for different stripes are distributed among the different storage units in a round robin manner.

The use of parity records as described in the Ouchi and Clark patents substantially reduces the cost of protecting data when compared to mirroring. However, while Ouchi and Clark teach a data recovery or protection means, they do not provide a means to keep a system operational to a user during data reconstruction. Normal operations are interrupted while a memory controller is powered down to permit a repair or replacement of the failed storage device, followed by a reconstruction of the data. Since this prior art relies exclusively on software for data reconstruction, the system can be disabled for a considerable time.

Prior art does not teach dynamic system recovery and continued operation without the use of duplicate or standby storage units. Mirroring requires a doubling of the number of storage units. A less extreme approach is the use of one or more standby units, i.e., additional spare disk drives which can be brought on line in the event any unit in the original set fails. Although this does not entail the cost of a fully mirrored system, it still requires additional storage units which otherwise serve no useful function.

It is therefore an object of the present invention to provide an enhanced method and apparatus for recovering from data loss in a computer system having multiple data storage units.

It is a further object of this invention to provide an enhanced method and apparatus whereby a computer system having multiple data storage units may continue to operate if one of the data storage units fails.

Another object of this invention is to reduce the cost of protecting data in a data processing system having multiple protected storage units.

A still further object of this invention is to increase the performance of a computer system having multiple data storage units when one of the data storage units fails and the system must reconstruct the data contained on the failed unit.

A storage controller services a plurality of data storage units. A storage management mechanism resident on the controller maintains parity records on the storage units it services. Data and parity blocks are organized as described in the patent to Clark et al. In the event of a storage unit failure, the system continues to operate. The storage management mechanism reconstructs data that was on the failed

unit as attempts are made to access that data, and stores it in the parity block areas of the remaining storage units.

The storage management mechanism includes a status map indicating, for each data block, the location of the corresponding parity block, and the status of the data block. If a storage unit fails, the storage management mechanism is placed in a failure operating mode. While in failure operating mode, the storage management mechanism checks the status map before accessing data on the failed storage unit. If the data has not yet been reconstructed, storage management must first reconstruct the data in that block of storage by successively reading and accumulating an Exclusive-OR (XOR) of the same blocks on all storage units in the parity group, including the parity block. The block of data resulting from this Exclusive-OR is the reconstructed data, which is then stored in the location of the parity block. The status map is then updated to indicate that the block has been reconstructed. Once the data has been reconstructed, it is only necessary to read from or write to the former parity block directly. In the same manner, storage management will reconstruct the data from a block of storage on the failed unit before writing to any other block on the same stripe (on a non-failed unit). This is required because the write operation to any block on the stripe will alter parity, making it impossible to later reconstruct the block of data on the failed unit. Thus, upon failure of a storage unit, system performance is initially degraded as read and write operations cause storage management to reconstruct data. As data is rebuilt, performance quickly improves.

In the preferred embodiment, the storage units are organized and parity information is generated and stored as described in the Clark et al. patent. Reconstructed data is stored in locations where parity data is normally stored for the stripe on which the lost data resided. There is no need to power down the storage controller or any other part of the system, repair the failed storage unit, and then reconstruct the lost data. In this preferred embodiment, the data are recovered and stored while a computer system using this storage management mechanism remains completely available to a user. The storage units operate without parity protection until the failed unit is repaired or replaced. This embodiment achieves continuous operation and single-level failure protection at very little additional cost.

In a first alternate embodiment, spare areas of storage in each non-failing storage unit are allocated to the reconstructed data. The total of these spare areas constitute a virtual spare storage unit. As data is reconstructed, it is placed in the virtual spare unit, and parity is maintained in the normal fashion. This alternative achieves an additional level of failure protection, because parity data continues to be maintained after a single storage unit failure. However, it may impose a need for additional storage space for the spare areas, or cause degraded performance if these spare areas are normally used for other purposes, such as temporary data storage.

In a second alternate embodiment, the storage management mechanism resides in the host system's operating software, but otherwise performs the same functions as a storage management mechanism residing on a storage controller. This embodiment will generally be slower than the preferred embodiment, but may reduce the cost of the storage controller.

Fig. 1 is a block diagram of a system incorporating the components of the preferred embodiment of this invention;

Fig. 2 is a diagram of a status map;

Fig. 3 is a flow diagram of the steps involved in a read operation during normal operating mode;

Fig. 4 is a flow diagram of the steps involved in transferring data to be written from the host to the storage controller;

Fig. 5 is a flow diagram of the steps involved in writing data to a storage device in normal operating mode;

Fig. 6 is a flow diagram of steps involved in read operations following a storage device failure.

Fig. 7 is a flow diagram of the steps involved in writing data to a storage device when a storage device has failed;

Fig. 8 is a block diagram of a system incorporating the components according to an alternative embodiment of this invention.

A block diagram of the major components of computer system 100 of the preferred embodiment of the present invention is shown in Figure 1. A host system 101, communicates over a bus 102 with a storage controller 103. Controller 103 comprises a programmed processor 104, non-volatile RAM 105, Exclusive-OR hardware 108, and cache memory (RAM) 109. Non-volatile RAM 105 contains a status map 106 and table of contents 107. Controller 103 controls the operation of storage units 121-124. In the preferred embodiment, units 121-124 are rotating magnetic disk storage units. While four storage units are shown in Fig. 1, it should be understood that the actual number of units attached to controller 103 is variable. It should also be understood that more than one controller 103 may be attached to host system 101. In the preferred embodiment, computer system 100 is an IBM AS/400 computer system, although any computer system could be used.

The storage area of each storage unit is divided into blocks 131-138. In the preferred embodiment, all storage units have identical storage capacity, and all parity protected blocks the same size. While it would be possible to employ this invention in configurations of varying sized storage units or varying sized blocks, the preferred embodiment simplifies the control

mechanism.

The set of all blocks located at the same location on the several storage units constitutes a stripe. In Fig. 1, storage blocks 131-134 constitute a first stripe, and blocks 135-138 constitute a second stripe. One of the blocks in each stripe is designated the parity block. Parity blocks 131,136 are shown shaded in Fig. 1. The remaining unshaded blocks 132-135,137-138 are data storage blocks for storing data. The parity block for the first stripe, consisting of blocks 131-134, is block 131. The parity block contains the Exclusive-OR of data in the remaining blocks on the same stripe.

In the preferred embodiment, parity blocks are distributed across the different storage units in a round robin manner, as shown in Fig. 1. Because with every write operation the system must not only update the block containing the data written to, but also the parity block for the same stripe, parity blocks are usually modified more frequently than data blocks. Distributing parity blocks among different storage units will in most cases improve performance by distributing the access workload. However, such distribution is not necessary to practicing this invention, and in an alternate embodiment it would be possible to place all parity blocks on a single storage unit.

In the preferred embodiment, one block of every stripe is dedicated to parity information. As an alternative embodiment, one of the stripes contains no parity protection. This stripe is reserved for temporary data which does not require protection. Fig. 8 shows this alternate embodiment in the stripe consisting of blocks 811-814. Because it is extra storage space not a part of the parity data protection scheme, this block may be of any arbitrary size.

The allocation of storage area into stripes as described above, each containing blocks of data and a parity block, is the same as that described in U.S. Patent 4,761,785 to Clark, et al., which is incorporated by reference.

Storage controller 103 includes programmed processor 104 executing a storage management program. The operation of the storage management program is described below. Controller 103 also includes hardware Exclusive-OR circuitry 108, for computing the Exclusive-OR of data in non-volatile RAM 105 or cache RAM 109. In an alternative embodiment, the Exclusive-OR operations could be performed by processor 104, but special hardware for this purpose will improve performance.

Non-volatile RAM 105 is used by controller 103 as a temporary queueing area for data waiting to be physically written to a storage unit. In addition to this temporary data, status map 106 and table of contents 107 are stored in non-volatile RAM 105. Table of contents 107 contains a mapping of the data waiting to be written to the location on which it is stored in the storage unit.

Status map 106 is used to identify the location of the corresponding parity block for each data block, and the status of each block of data during failure recovery mode. Status map 106 is shown in detail in Fig. 2. It contains a separate table of status map entries for each storage unit. Each status map entry 201 contains the location 202 of a block of data on the storage unit, a status bit 203 indicating whether or not the data needs to be recovered when operating in failure mode, and the location of the corresponding parity block 204.

Referring again to Fig. 1, cache memory 109 is a volatile random access memory that is used to store data read from a storage unit. It serves as a buffer when transferring data from a storage unit to host system 101 in a read operation. In addition, data is saved in cache 109 in response to indications from the host system 101 that the data has a high probability of modification and re-writing. Because unmodified data must be exclusive-ORed with modified data to update the corresponding parity data, saving read data in cache 109 can eliminate the need to read it again immediately before a write operation. Cache 109 exists only to improve performance. In an alternative embodiment, it would be possible to practice this invention without it. Cache 109 is identified as a volatile RAM because it is not necessary to the integrity of the system that data read from storage be preserved in non-volatile memory. However, the cache could be implemented as part of the non-volatile memory 105. Depending on the relative cost and size of memory modules, such an approach may be desirable.

The function of the system in conjunction with the hardware and software features necessary to this invention is described below. The system has two operating modes: normal and failure mode. The system operates in normal mode when all disk storage devices are functioning properly. When one storage device fails, the mode of operation changes to failure mode, but the system continues to operate.

A READ operation in normal mode is shown in Fig. 3. The READ operation is performed by accepting a READ command from the host at step 301, and determining whether the data requested exists in non-volatile RAM 105 or cache 109 at step 302. If so, the data in non-volatile RAM or cache is sent directly to the host at step 304. Otherwise, data is first read from the appropriate storage unit into the cache 109 at step 303, and from there transferred to the host system at step 304. The cache 109 also improves performance during WRITE operations. If the original version of data to be updated is already in cache 109 when a WRITE operation is processed, it is not necessary to read the data again in order to update parity, thus improving system performance. The contents of cache 109 are managed using any of various cache management techniques known in the art.

A WRITE operation is performed by two asynchronous tasks running in the storage controller's processor 104. One task communicates with the host via bus 102, and is shown in Fig. 4. The WRITE operation begins when it accepts a WRITE command from the host at step 401. It then checks table of contents 107 to determine whether sufficient space is available in non-volatile RAM 105 to store the data to be written to storage in step 402 (Note that space available includes space used by back-level versions of the data to be written, as well as unused space). If space is not available, controller 103 can not receive data from the host, and must wait for space to become available at step 403 (i.e., it must wait for data already in non-volatile RAM 105 to be written to storage 121-124). When space becomes available in non-volatile RAM 105, data is copied from host 101 into non-volatile RAM 105, and table of contents 107 is updated at step 404. Processor 104 then issues an operation complete message to the host at step 405. Upon receipt of the operation complete message, the host is free to continue processing as if the data were actually written to storage 121-124, although in fact the data may wait awhile in non-volatile RAM 105. From the host's perspective, the operation will appear to be complete.

The second asynchronous task writes data from non-volatile RAM 105 to a storage unit. A flow diagram of this task in normal mode is shown in Fig. 5. The task selects a WRITE operation from among those queued in non-volatile RAM at step 501. The selection criteria are not a part of this invention, and could be, e.g., First-in-first-out, Last-in-first-out, or some other criteria based on system performance and other considerations. When the WRITE operation is performed, parity must be updated. By taking the Exclusive-OR of the new write data with the old data, it is possible to obtain a bit map of those bits being changed by the WRITE operation. Exclusive-ORing this bit map with the existing parity data produces the updated parity data. Therefore, before writing to storage, the task first checks whether the old data exists in the cache 109 in unmodified form at step 502. If not, it is read into the cache from storage at step 503. This old data in the cache is then Exclusive-ORed with the new data in non-volatile RAM to produce the bit map of changed data at step 504. The bit map is saved temporarily in non-volatile RAM 105 while the new data is written to one of the storage devices 121-124. The old parity data is then read into the cache (if not already there) at steps 506,507, and Exclusive-ORed with the bit map to produce the new parity data at step 508. This new parity data is written to one of the storage devices 121-124 and the table of contents is updated at step 509, completing the WRITE operation.

When a storage unit failure is detected, the system begins operating in failure mode. The failure of a storage unit means failure to function, i.e., to access data. Such a failure is not necessarily caused by a breakdown of the unit itself. For example, the unit could be powered off, or a data cable may be disconnected. From the perspective of the system, any such failure, whatever the cause, is a failure of the storage unit. Detection mechanisms which detect such failures are known in the art. Common mechanisms include a time-out after not receiving a response, and continued high error rates in received data.

Figure 6 illustrates the READ operation when the system is operating in failure mode. As in the case of normal mode READ operations, when a READ is accepted from the host at step 601, the controller first checks its non-volatile RAM 105 and its volatile cache 109 for the desired data at step 602. If the data exists in non-volatile RAM or cache, the data is transferred to the host via system bus 102. If the data is not in non-volatile RAM or cache, and resides on a storage device which has not failed (step 603), the data is read into the cache from the storage device in the normal manner at step 604. If the data resides on a failed storage unit, the controller checks the status map entry 201 in status map 106 for the location in storage of the desired data at step 605. The status map entry will indicate whether the data has been recovered, i.e., whether it has been reconstructed by exclusive-ORing and stored at some alternate location. If the status map indicates that the data has not been recovered (step 605) the controller successively reads the corresponding locations on all storage units except the failing one at step 608. Each block of data read is XORed by the XOR hardware 108 with the accumulated XOR results of the previously read blocks. The final XOR results constitute the reconstructed data of the failed device. This reconstructed data is written to the parity block corresponding to this block of data at step 609. The location of this block is stored in a parity block address field 204 of the status map 108. After writing the recovered data to the parity block location, status map 108 is updated at step 610 by changing the status bit 203 of each block in the same stripe to a '1' to indicate that the data has been recovered. The reconstructed data is sent to the host at step 611. If the status bit 203 originally contained a '1', indicating that data had been recovered, the controller would obtain the location of the former parity block area (where recovered data is stored) from the status map at step 606, and read the data from this location directly into the cache at step 607. By this device, it is only necessary to read all disk storage units once to recover any particular block of data. Once recovered, the physical storage location of that data is effectively relocated to the location that was formerly used for parity storage, and any subsequent reads of that block need only read the one storage unit.

Figure 7 illustrates the write to storage operation when the system is operating in failure mode. As with the normal mode WRITE, a host communications task shown in Fig. 4 receives data to be written from the host via bus 102. The write to storage task selects a write operation from the queue in non-volatile RAM 105 at step 701. The controller determines whether the data is to be written to a failed unit (step 702) and checks the status map (steps 703, 709). If the data is to be written to a failing unit, and the data in the block has not yet been recovered, the block must be recovered before any write operations are possible. Recovery follows the same steps described above for a READ operation. Each block in the same stripe of blocks (including the parity block) is read in turn, and its contents Exclusive-ORed with the cumulative Exclusive-OR of the previously read blocks at step 704. The result, which is the reconstructed data, is written to the location used for the parity block at step 705. Once the recovery of the entire block is complete, the new data (which would typically encompass only a portion of the block) is written over the recovered data in the former parity location at step 706, and the status map updated to indicate that the block has been recovered at step 707. If data is to be written to a failing unit, but the data has already been recovered, it is written directly to the former parity location, now used for storage of recovered data, at step 708.

If data is being written to a non-failing unit when operating in failure mode, the controller checks the status map at step 709. If the status is '1', indicating that the block of data in the same stripe on the failing unit has already been recovered, the WRITE data is written directly to the non-failing storage unit at step 710. If the status is '0', data can not be directly written to the non-failing unit, because such an operation would alter parity, making it impossible to later reconstruct the corresponding data in the failed unit. Accordingly, in the preferred embodiment, the controller will first recover the block of data in the same stripe on the failing unit. As shown if Fig. 7, the block of data in the failing unit is first reconstructed by Exclusive-ORing at step 711, and saved in the parity block location at step 712, following the steps described above. The WRITE data is then written to its storage unit at step 713, and the status map is updated at step 714. Note that if the parity block for the stripe containing the data to be written is on the failing unit, no reconstruction is necessary, since parity will be lost anyway. Therefore, the status for all blocks on this stripe is set to 1 when the storage unit failure is detected. The effect will be to cause data on this stripe to be directly written to storage as if the corresponding block on the failing unit had already been recovered. For example, referring to Fig. 1, if storage unit 121 fails, the controller will immediately set the status of blocks 132-134 to '1', so that WRITE operations to these blocks can proceed directly. In an alternative embodiment, if the WRITE operation is to a non-failing unit, and the corresponding block on the failing unit has not been recovered, it would be possible to follow the same steps used for a normal mode WRITE operation to update the parity block, preserving the ability to reconstruct the failing unit's data later if a READ or WRITE of the data on the failed unit is requested.

In the preferred embodiment, parity blocks are used to store reconstructed data, with the result that the system runs without parity protection after a single storage unit failure. An alternative embodiment is possible where a sufficiently large spare storage stripe or stripes is reserved on the storage units, as shown in Fig. 8. This spare storage stripe might contain temporary data which does not require parity protection and which can be overwritten if the need arises, or it might contain no data at all. In this alternative embodiment, reconstructed data is relocated to a block of a spare storage stripe 811-814 instead of the parity block. This alternative is only possible where sufficient spare storage exists to accommodate the non-spare contents of the failed unit. It would also have the consequence of reducing the amount of temporary storage available to the system, possibly degrading performance or reducing the number of users the system can service. In this alternative embodiment, normal mode READ and WRITE operations are performed in exactly the same manner as in the preferred embodiment. When operating in failure mode, the status map is checked, and the data reconstructed as needed, in the manner described above. However, instead of writing the reconstructed data to the parity block, it is written to a block in spare storage. Another field is required in status map 106 to record the new location of the data which was contained on the failed unit. In addition, with any WRITE operation parity is updated in the same manner as a WRITE operation in normal mode. This is done after any reconstruction of data on the failed unit.

In another alternative embodiment, parity protection and mirroring are combined on the same system. Some of the data contained on the storage units is protected by the parity protection mechanism described herein, while other data is mirrored. In the event of a storage unit failure, the parity protected data is reconstructed and stored as described above, while the mirrored data is accessed from the storage unit containing the mirrored copy.

Although a specific embodiment of the invention has been disclosed along with certain alternatives, it will be recognized by those skilled in the art that additional variations in form and detail may be made. In particular, while the disclosed preferred embodiment employs magnetic disk storage units, the invention is applicable to other storage device technologies having erasable, read/write

characteristics.

Claims

1. A method of operating a computer system having a stripe of storage blocks, said stripe comprising a plurality of data storage blocks for containing data and a parity storage block for containing the parity of the data stored in said data storage blocks, each of said storage blocks being contained on a respective data storage unit, said method comprising the steps of:

   reconstructing data contained in a storage block, while the data storage unit containing said storage block is failing, from the remaining storage blocks in the stripe; and

   storing data reconstructed by said reconstructing step on one of said data storage units.

2. The method of operating a computer system of claim 1, wherein said reconstructing data step reconstructs data when attempts are made to access said data.

3. The method of operating a computer system of claim 1 or 2, wherein said storing data step stores the reconstructed data in said parity storage block.

4. The method of operating a computer system of claim 3, wherein said reconstructing data step reconstructs data when attempts are made to access said data.

5. The method of operating a computer system of any one of claims 1-4, wherein said data storage units contain a spare storage block, and said storing data step stores the reconstructed data in said spare storage block.

6. A storage apparatus for a computer system, comprising:

   at least three data storage units;

   at least one stripe of storage blocks, each stripe comprising a plurality of data storage blocks for containing data and a parity storage block for containing the parity of the data stored in said data storage blocks, each of said storage blocks being contained on a respective data storage unit;

   means for reconstructing the data contained in one of said data storage blocks, while the data storage unit containing said block is failing, from the remaining storage blocks in the stripe; and

   means for storing said reconstructed data on one of said data storage units.

7. The storage apparatus for the computer system of claim 6, wherein said means for storing said reconstructed data stores said data in said parity storage block.

8. The storage apparatus for the computer system of claim 7, wherein said means for reconstructing the data comprises a storage controller, said storage controller comprising:

   a programmable processor executing a storage management program; and

   a non-volatile random access memory.

9. The storage apparatus for the computer system of claim 7,

   wherein said data processing system comprises at least two of said stripes of storage blocks; and

   wherein said parity storage blocks are distributed among said data storage units in a round robin manner.

10. The storage apparatus for the computer system of claim 7, wherein each of said data storage units is a rotating magnetic disk drive storage unit.

11. The storage apparatus for the computer system of anyone of claims 6-10,

   wherein each of said data storage units contains a spare storage block; and

   wherein said means for storing said reconstructed data stores said data in one of said spare storage blocks.

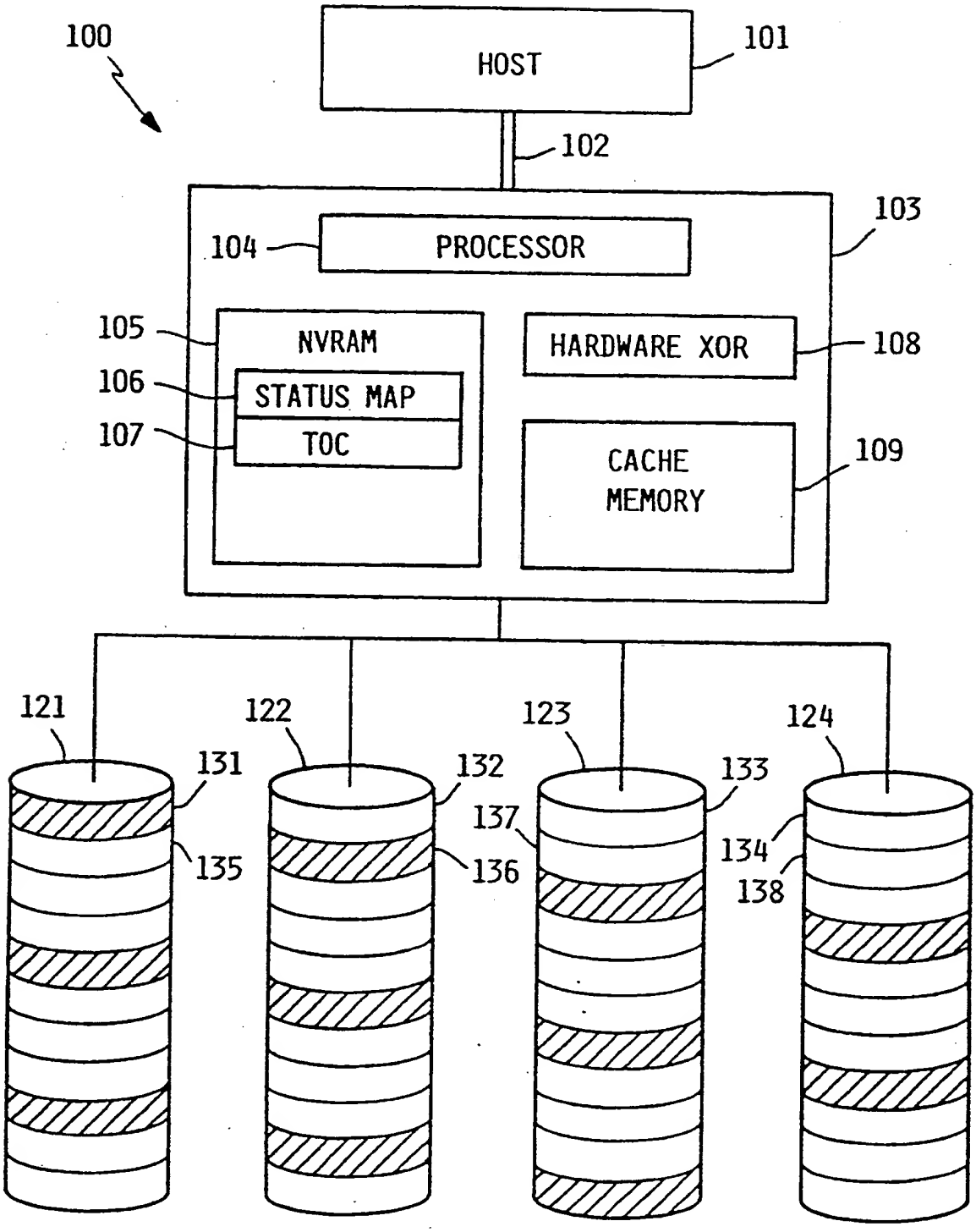
100
HOST
101
102
103
104
PROCESSOR
105
NVRAM
106
STATUS MAP
107
TOC
HARDWARE XOR
108
CACHE MEMORY
109
121
122
123
124
131
132
133
135
136
137
134
138

FIG. 1

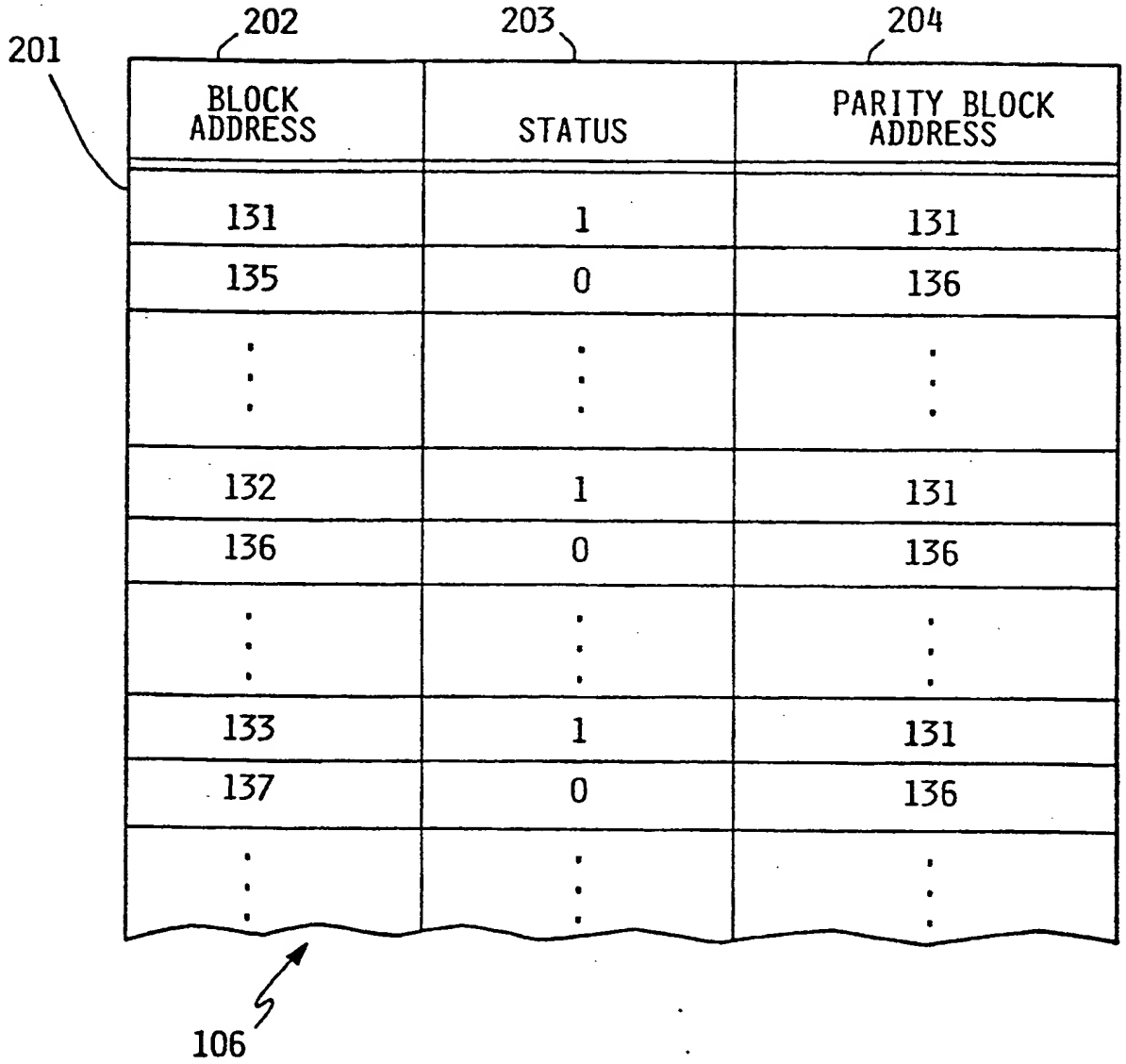
201
202
203
204
BLOCK ADDRESS
STATUS
PARITY BLOCK ADDRESS
131
1
131
135
0
136
132
1
131
136
0
136
133
1
131
137
0
136
106

FIG. 2

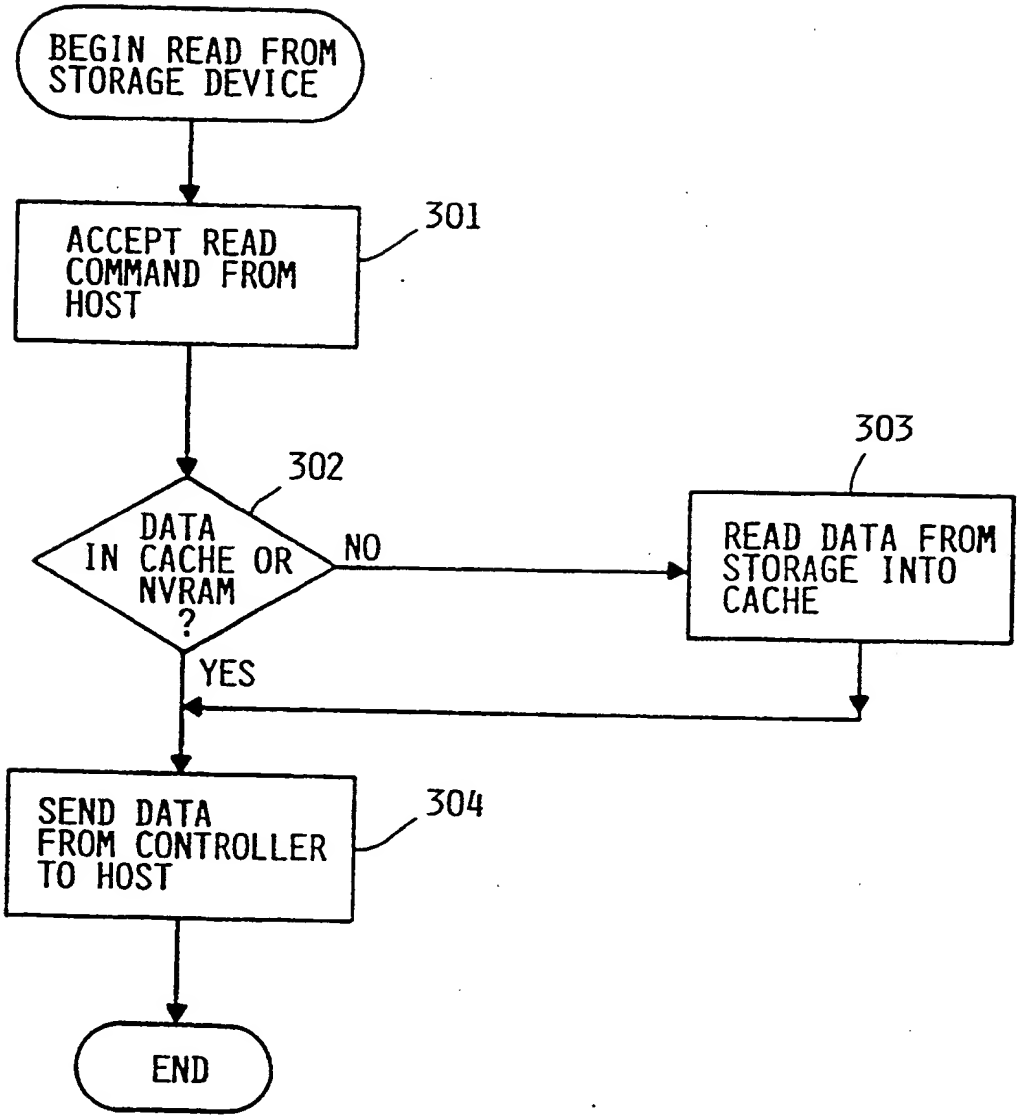
BEGIN READ FROM STORAGE DEVICE
ACCEPT READ COMMAND FROM HOST
301
DATA IN CACHE OR NVRAM ?
302
NO
YES
READ DATA FROM STORAGE INTO CACHE
303
SEND DATA FROM CONTROLLER TO HOST
304
END

FIG. 3

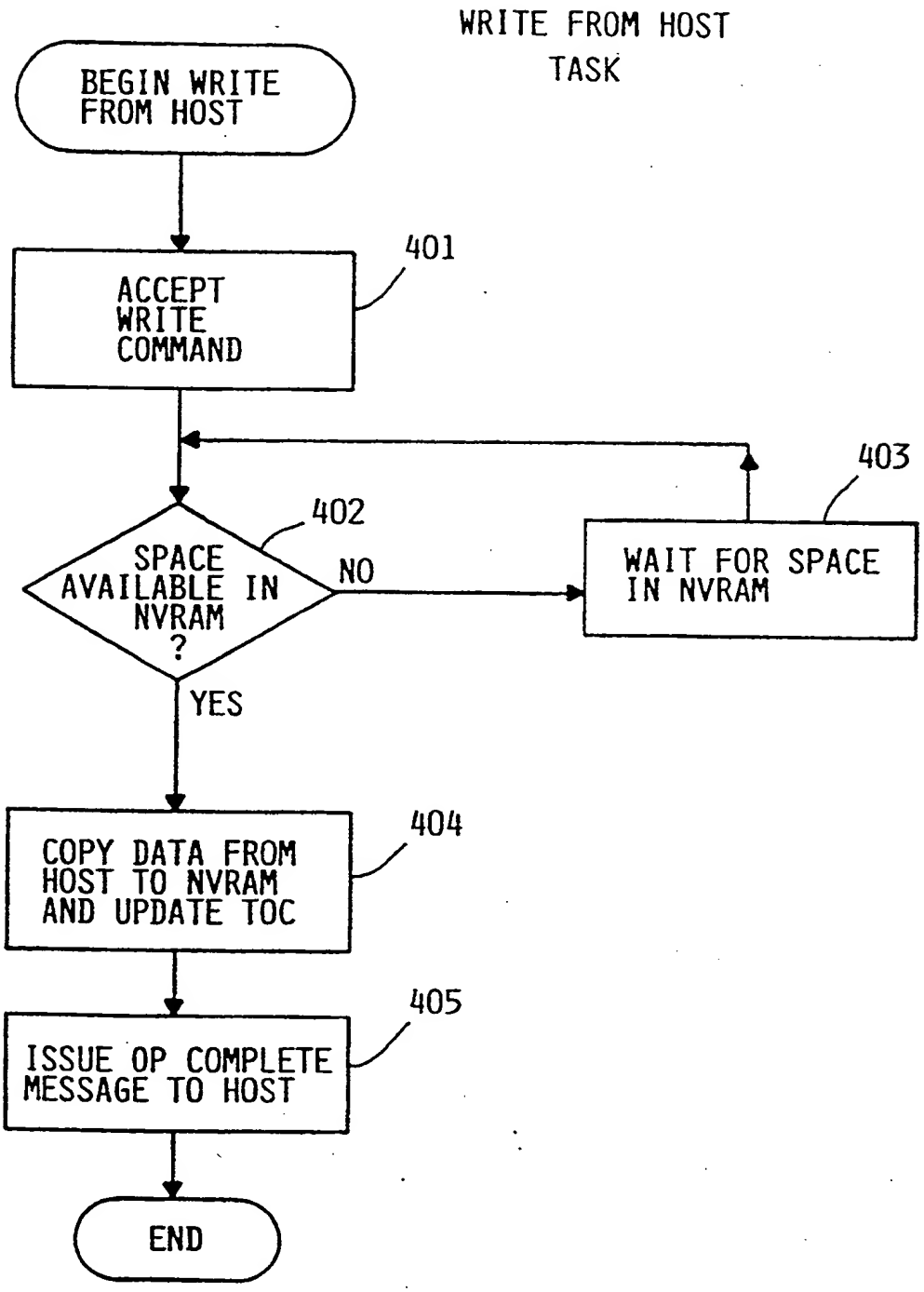
WRITE FROM HOST TASK
BEGIN WRITE FROM HOST
ACCEPT WRITE COMMAND
401
SPACE AVAILABLE IN NVRAM ?
402
NO
WAIT FOR SPACE IN NVRAM
403
YES
COPY DATA FROM HOST TO NVRAM AND UPDATE TOC
404
ISSUE OP COMPLETE MESSAGE TO HOST
405
END

FIG. 4

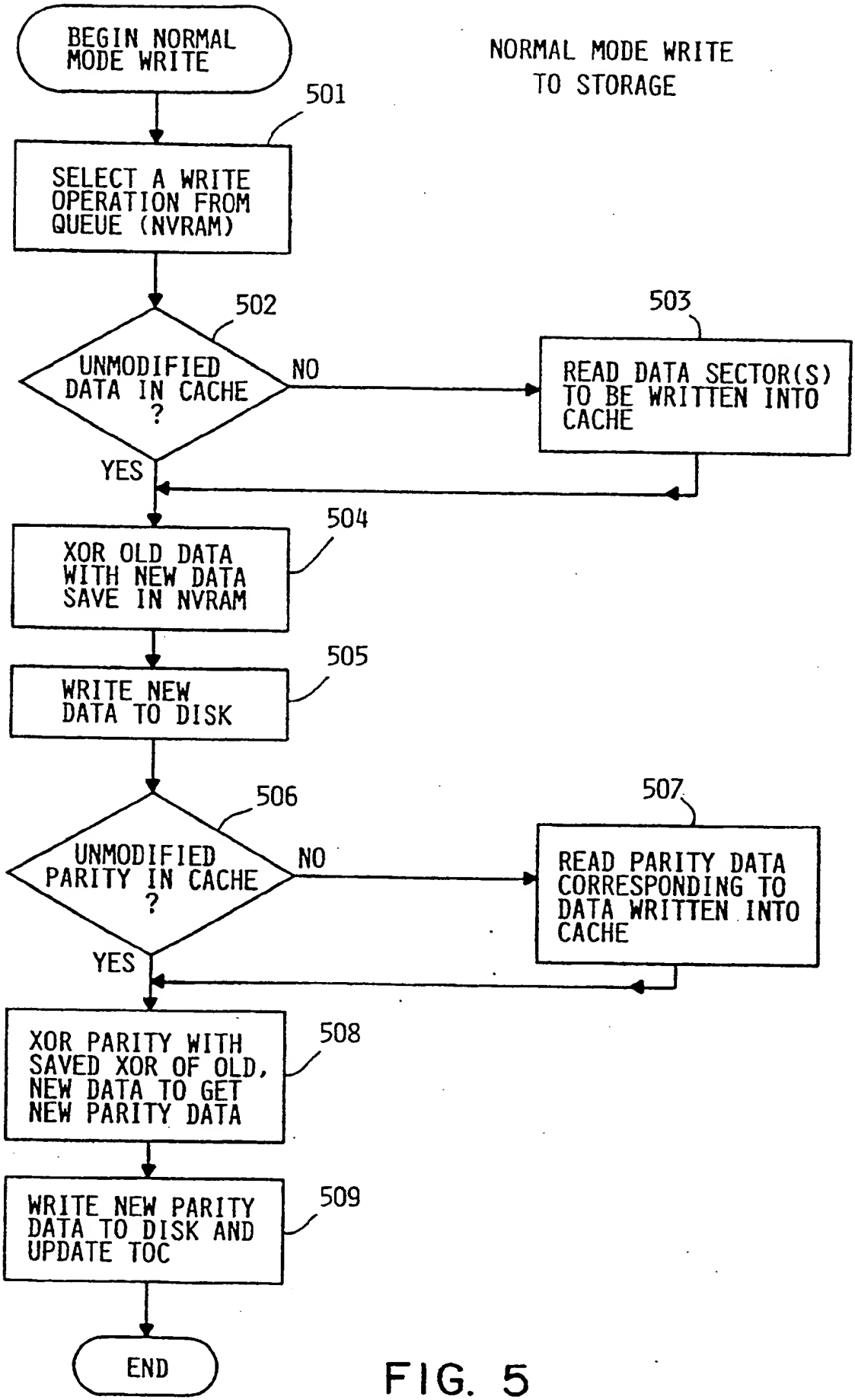
NORMAL MODE WRITE TO STORAGE
BEGIN NORMAL MODE WRITE
501
SELECT A WRITE OPERATION FROM QUEUE (NVRAM)
502
UNMODIFIED DATA IN CACHE ?
NO
503
READ DATA SECTOR(S) TO BE WRITTEN INTO CACHE
YES
504
XOR OLD DATA WITH NEW DATA SAVE IN NVRAM
505
WRITE NEW DATA TO DISK
506
UNMODIFIED PARITY IN CACHE ?
NO
507
READ PARITY DATA CORRESPONDING TO DATA WRITTEN INTO CACHE
YES
508
XOR PARITY WITH SAVED XOR OF OLD, NEW DATA TO GET NEW PARITY DATA
509
WRITE NEW PARITY DATA TO DISK AND UPDATE TOC
END

FIG. 5

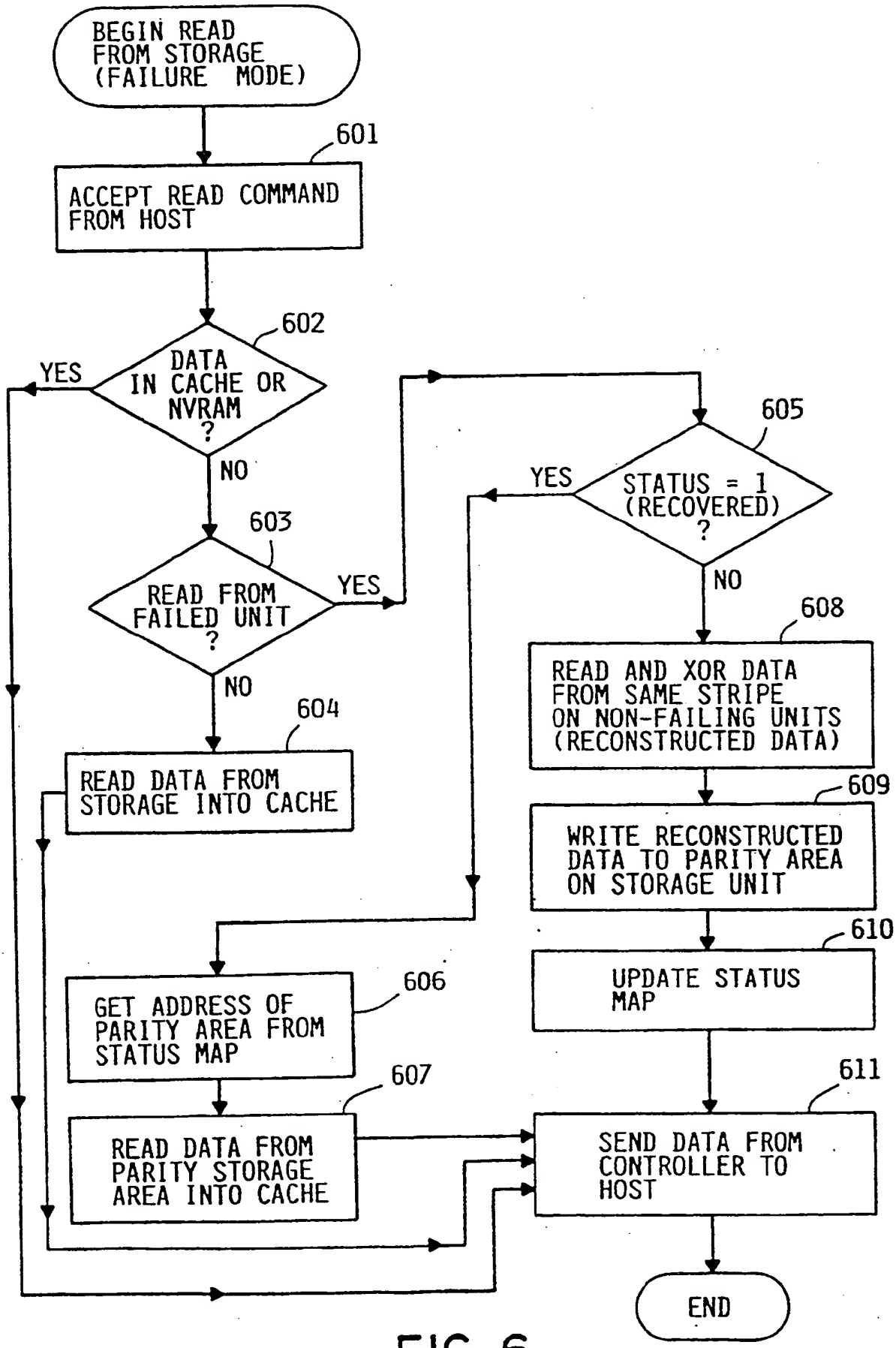
BEGIN READ FROM STORAGE (FAILURE MODE)
601
ACCEPT READ COMMAND FROM HOST
602
DATA IN CACHE OR NVRAM ?
YES
NO
603
READ FROM FAILED UNIT ?
YES
NO
604
READ DATA FROM STORAGE INTO CACHE
605
STATUS = 1 (RECOVERED) ?
YES
NO
608
READ AND XOR DATA FROM SAME STRIPE ON NON-FAILING UNITS (RECONSTRUCTED DATA)
609
WRITE RECONSTRUCTED DATA TO PARITY AREA ON STORAGE UNIT
610
UPDATE STATUS MAP
606
GET ADDRESS OF PARITY AREA FROM STATUS MAP
607
READ DATA FROM PARITY STORAGE AREA INTO CACHE
611
SEND DATA FROM CONTROLLER TO HOST
END

FIG. 6

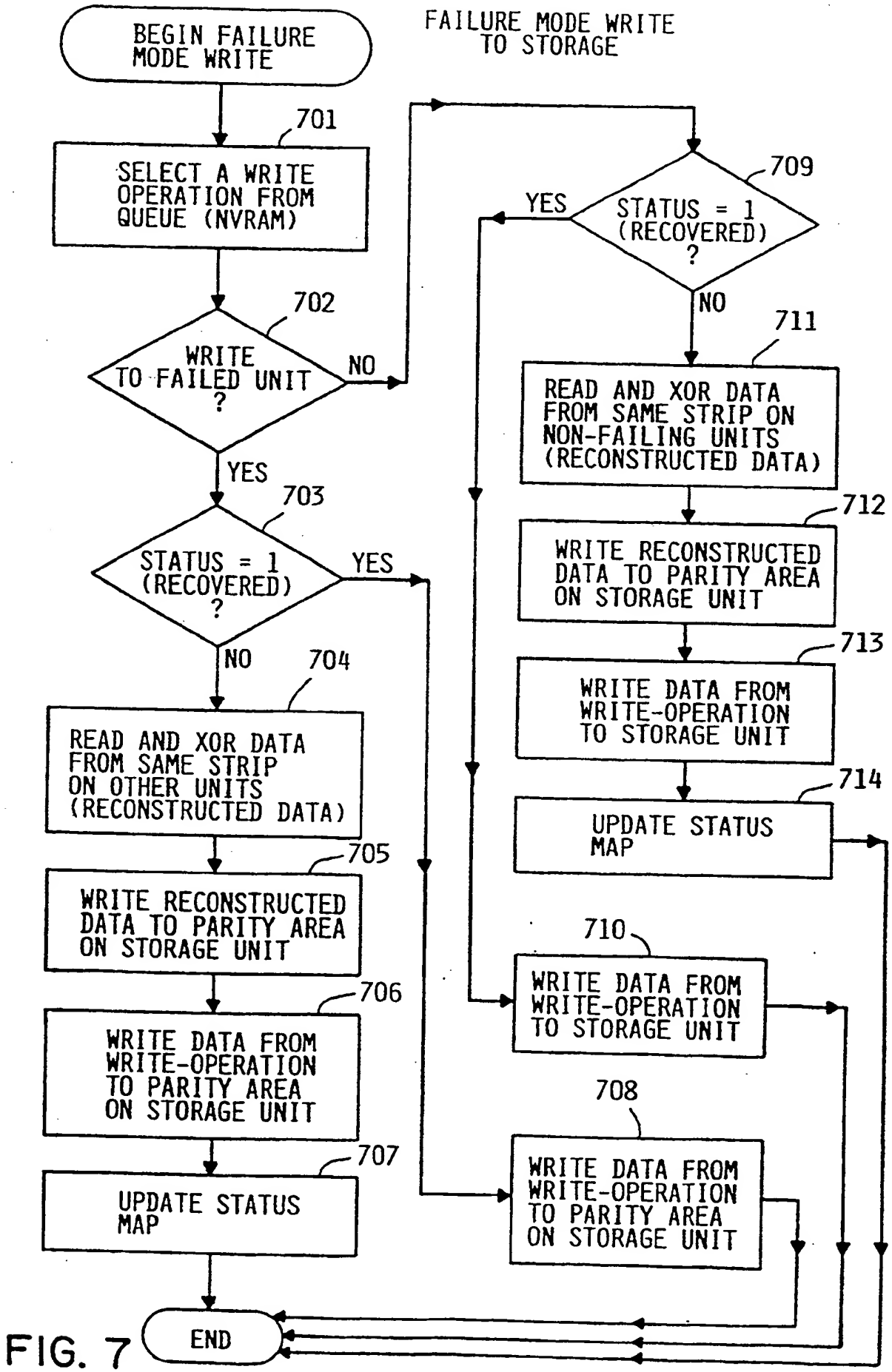
FAILURE MODE WRITE TO STORAGE
BEGIN FAILURE MODE WRITE
701
SELECT A WRITE OPERATION FROM QUEUE (NVRAM)
702
WRITE TO FAILED UNIT ?
NO
YES
703
STATUS = 1 (RECOVERED) ?
YES
NO
704
READ AND XOR DATA FROM SAME STRIP ON OTHER UNITS (RECONSTRUCTED DATA)
705
WRITE RECONSTRUCTED DATA TO PARITY AREA ON STORAGE UNIT
706
WRITE DATA FROM WRITE-OPERATION TO PARITY AREA ON STORAGE UNIT
707
UPDATE STATUS MAP
709
STATUS = 1 (RECOVERED) ?
YES
NO
711
READ AND XOR DATA FROM SAME STRIP ON NON-FAILING UNITS (RECONSTRUCTED DATA)
712
WRITE RECONSTRUCTED DATA TO PARITY AREA ON STORAGE UNIT
713
WRITE DATA FROM WRITE-OPERATION TO STORAGE UNIT
714
UPDATE STATUS MAP
710
WRITE DATA FROM WRITE-OPERATION TO STORAGE UNIT
708
WRITE DATA FROM WRITE-OPERATION TO PARITY AREA ON STORAGE UNIT
END

FIG. 7

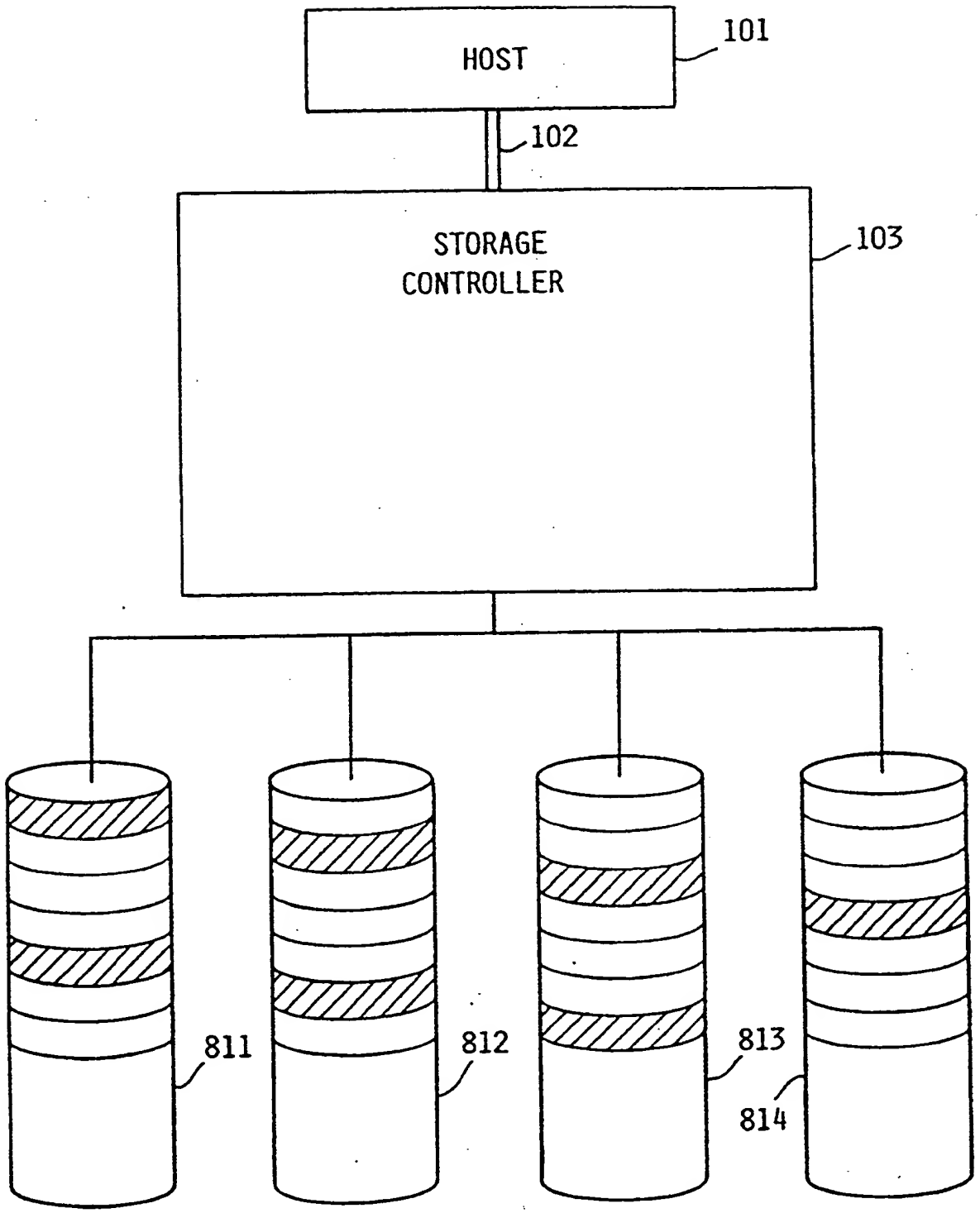

FIG. 8